В данном разделе рассматривается альтернативный подход к разработке алгоритмов, известный как метод разбиения (“разделяй и властвуй”). Разработаем с помощью этого подхода алгоритм сортировки, время работы которого в наихудшем случае намного меньше времени работы алгоритма, работающего по методу включений. Одно из преимуществ алгоритмов, разработанных методом разбиения, заключается в том, что время их работы часто легко определяется с помощью технологий, описанных в главе 4.

2.3.1 Метод декомпозиции

Многие полезные алгоритмы имеют ***рекурсивную*** структуру: для решения данной задачи они рекурсивно вызывают сами себя один или несколько раз, чтобы решить вспомогательную задачу, имеющую непосредственное отношение к поставленной задаче. Такие алгоритмы зачастую разрабатываются с помощью метода ***декомпозиции***, или ***разбиения***: сложная задача разбивается на несколько более простых, которые подобны исходной задаче, но имеют меньший объем; далее эти вспомогательные задачи решаются рекурсивным методом, после чего полученные решения комбинируются с целью получить решение исходной задачи.

Парадигма, лежащая в основе метода декомпозиции “разделяй и властвуй”, на каждом уровне рекурсии включает в себя три этапа.

Разделение задачи на несколько подзадач.

Покорение — рекурсивное решение этих подзадач. Когда объем подзадачи достаточно мал, выделенные подзадачи решаются непосредственно.

Комбинирование решения исходной задачи из решений вспомогательных задач.

Алгоритм ***сортировки слиянием*** (merge sort) в большой степени соответствует парадигме метода разбиения. На интуитивном уровне его работу можно описать таким образом.

Разделение: сортируемая последовательность, состоящая из n элементов, разбивается на две меньшие последовательности, каждая из которых содержит $n/2$ элементов.

Покорение: сортировка обеих вспомогательных последовательностей методом слияния.

Комбинирование: слияние двух отсортированных последовательностей для получения окончательного результата.

Рекурсия достигает своего нижнего предела, когда длина сортируемой последовательности становится равной 1. В этом случае вся работа уже сделана, поскольку любую такую последовательность можно считать упорядоченной.

Основная операция, которая производится в процессе сортировки по методу слияний, — это объединение двух отсортированных последовательностей в ходе комбинирования (последний этап). Это делается с помощью вспомогательной процедуры MERGE(A,p,q,r), где A — массив, а p, q и r — индексы, нумерующие элементы массива, такие, что $p \leqslant q < r$. В этой процедуре предполагается, что элементы подмассивов $A[p..q]$ и $A[q+1..r]$ упорядочены. Она ***сливает*** эти два подмассива в один отсортированный, элементы которого заменяют текущие элементы подмассива $A[p..r]$.

Для выполнения процедуры MERGE требуется время $\Theta(n)$, где $n = r-p+1$ — количество подлежащих слиянию элементов. Процедура работает следующим образом. Возвращаясь к наглядному примеру сортировки карт, предположим, что на столе лежат две стопки карт, обращенных лицевой стороной вниз. Карты в каждой стопке отсортированы, причем наверху находится карта наименьшего достоинства. Эти две стопки нужно объединить в одну выходную, в которой карты будут рассортированы и также будут обращены рубашкой вверх. Основной шаг состоит в том, чтобы из двух младших карт выбрать самую младшую, извлечь ее из соответствующей стопки (при этом в данной стопке верхней откажется новая карта) и поместить в выходную стопку. Этот шаг повторяется до тех пор, пока в одной из входных стопок не кончатся карты, после чего оставшиеся в другой стопке карты нужно поместить в выходную стопку. С вычислительной точки зрения выполнение каждого основного шага занимает одинаковые промежутки времени, так как все сводится к сравнению достоинства двух верхних карт. Поскольку необходимо выполнить, по крайней мере, n основных шагов, время работы процедуры слияния равно $\Theta(n)$.

Описанная идея реализована в представленном ниже псевдокоде, однако в нем также есть дополнительное ухищрение, благодаря которому в ходе каждого основного шага не приходится проверять, является ли каждая из двух стопок пустой. Идея заключается в том, чтобы поместить в самый низ обеих объединяемых колод так называемую ***сигнальную*** карту особого достоинства, что позволяет упростить код. Для обозначения сигнальной карты используется символ ∞. Не существует карт, достоинство которых больше достоинства сигнальной карты. Процесс продолжается до тех пор, пока проверяемые карты в обеих стопках не окажутся сигнальными. Как только это произойдет, это будет означать, что все несигнальные карты уже помещены в выходную стопку. Поскольку заранее известно, что в выходной стопке должно содержаться ровно $r-p+1$ карта, выполнив такое количество основных шагов, можно остановиться:

MERGE(A,p,q,r)

$n_1 \leftarrow q-p+1$

$n_2 \leftarrow r-q$

Создаем массивы $L[1..n_1+1]$ и $R[1..n_2+1]$

```
4  for i ← 1 to n1
5      do L[i] ← A[p + i − 1]
6  for j ← 1 to n2
7      do R[j] ← A[q + j]
8  L[n1 + 1] ← ∞
9  R[n2 + 1] ← ∞
10 i ← 1
11 j ← 1
12 for k ← p to r
13     do if L[i] ⩽ R[j]
14           then A[k] ← L[i]
15                i ← i + 1
16           else A[k] ← R[j]
17                j ← j + 1
```

Подробно опишем работу процедуры MERGE. В строке 1 вычисляется длина n_1 подмассива $A[p..q]$, а в строке 2 — длина n_2 подмассива $A[q+1..r]$. Далее в строке 3 создаются массивы L ("левый" — "left") и R ("правый" — "right"), длины которых равны $n_1 + 1$ и $n_2 + 1$ соответственно. В цикле **for** в строках 4 и 5 подмассив $A[p..q]$ копируется в массив $L[1..n_1]$, а в цикле **for** в строках 6 и 7 подмассив $A[q+1..r]$ копируется в массив $R[1..n_2]$. В строках 8 и 9 последним элементам массивов L и R присваиваются сигнальные значения.

Как показано на рис. 2.3, в результате копирования и добавления сигнальных карт получаем массив L с последовательностью чисел $\langle 2, 4, 5, 7, \infty \rangle$ и массив R с последовательностью чисел $\langle 1, 2, 3, 6, \infty \rangle$. Светло-серые ячейки массива A содержат конечные значения, а светло-серые ячейки массивов L и R — значения, которые еще только должны быть скопированы обратно в массив A. В светло-серых ячейках находятся исходные значения из подмассива $A[9..16]$ вместе с двумя сигнальными картами. В темно-серых ячейках массива A содержатся значения, которые будут заменены другими, а в темно-серых ячейках массивов L и R — значения, уже скопированные обратно в массив A. В частях рисунка *а–з* показано состояние массивов A, L и R, а также соответствующие индексы k, i и j перед каждой итерацией цикла в строках 12–17. В части *и* показано состояние массивов и индексов по завершении работы алгоритма. На данном этапе подмассив $A[9..16]$ отсортирован, а два сигнальных значения в массивах L и R — единственные элементы, оставшиеся в этих массивах и не скопированные в массив A. В строках 10–17, проиллюстрированных на рис. 2.3, выполняется $r - p + 1$ основных шагов, в ходе каждого из которых производятся манипуляции с инвариантом цикла, описанным ниже.

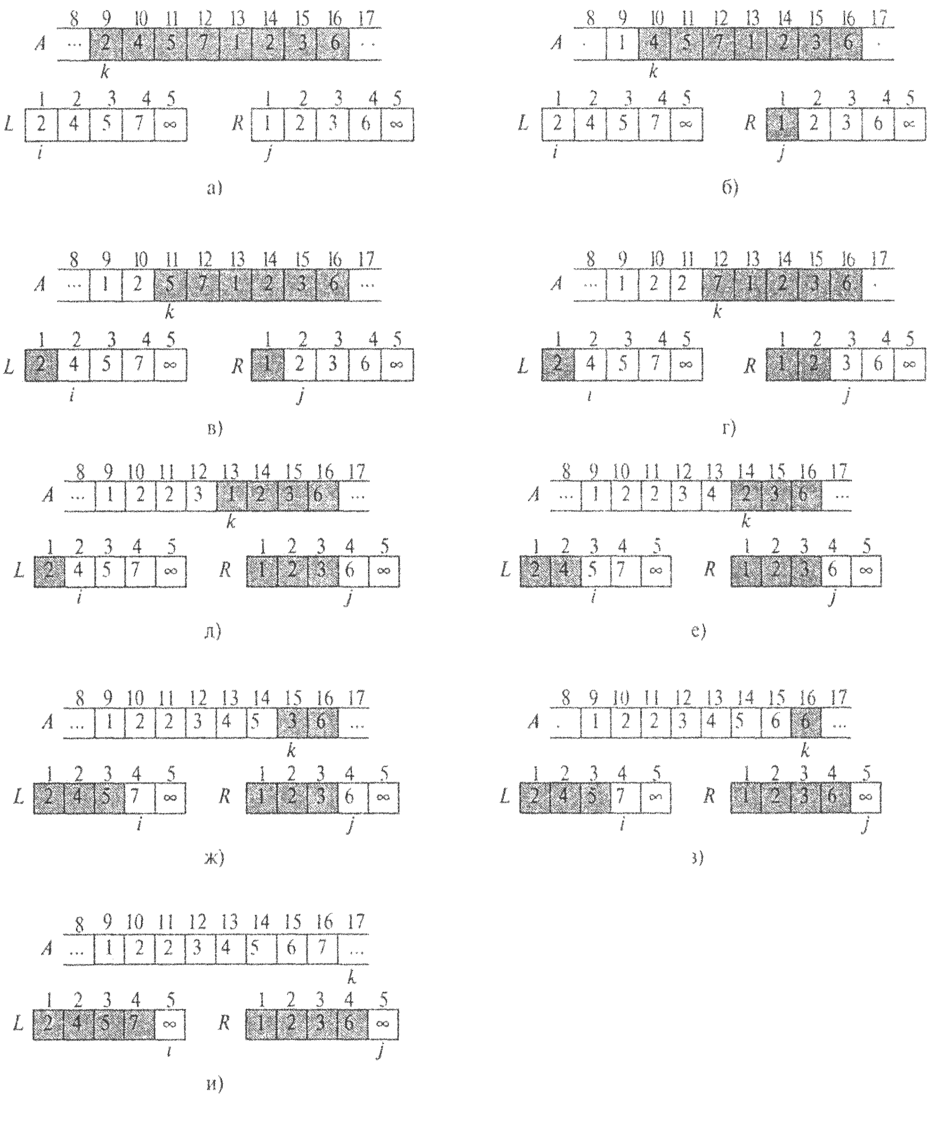

Рис. 2.3. Операции, выполняемые в строках 10–17 процедуры MERGE($A, 9, 12, 16$), когда в подмассиве $A[9..16]$ содержится последовательность $\langle 2,4,5,7,1,2,3,6 \rangle$

Перед каждой итерацией цикла **for** в строках 12–17, подмассив $A[p..k-1]$ содержит $k-p$ наименьших элементов массивов $L[1..n_1+1]$ и $R[1..n_2+1]$ в отсортированном порядке. Кроме того, элементы $L[i]$ и $R[i]$ являются наименьшими элементами массивов L и R, которые еще не скопированы в массив A.

Необходимо показать, что этот инвариант цикла соблюдается перед первой итерацией рассматриваемого цикла **for**, что каждая итерация цикла не нарушает его, и что с его помощью удается продемонстрировать корректность алгоритма, когда цикл заканчивает свою работу.

Инициализация. Перед первой итерацией цикла $k = p$, поэтому подмассив $A[p..k-1]$ пуст. Он содержит $k - p = 0$ наименьших элементов массивов L и R. Поскольку $i = j = 1$, элементы $L[i]$ и $R[j]$ — наименьшие элементы массивов L и R, не скопированные обратно в массив A.

Сохранение. Чтобы убедиться, что инвариант цикла сохраняется после каждой итерации, сначала предположим, что $L[i] \leqslant R[j]$. Тогда $L[i]$ — наименьший элемент, не скопированный в массив A. Поскольку в подмассиве $A[p..k-1]$ содержится $k-p$ наименьших элементов, после выполнения строки 14, в которой значение элемента $L[i]$ присваивается элементу $A[k]$, в подмассиве $A[p..k]$ будет содержаться $k - p + 1$ наименьший элемент. В результате увеличения параметра k цикла **for** и значения переменной i (строка 15), инвариант цикла восстанавливается перед следующей итерацией. Если же выполняется неравенство $L[i] > R[j]$, то в строках 16 и 17 выполняются соответствующие действия, в ходе которых также сохраняется инвариант цикла.

Завершение. Алгоритм завершается, когда $k = r + 1$. В соответствии с инвариантом цикла, подмассив $A[p..k-1]$ (т.е. подмассив $A[p..r]$) содержит $k - p = r - p + 1$ наименьших элементов массивов $L[1..n_1 + 1]$ и $R[1..n_2 + 1]$ в отсортированном порядке. Суммарное количество элементов в массивах L и R равно $n_1 + n_2 + 2 = r - p + 3$. Все они, кроме двух самых больших, скопированы обратно в массив A, а два оставшихся элемента являются сигнальными.

Чтобы показать, что время работы процедуры MERGE равно $\Theta(n)$, где $n = r - p + 1$, заметим, что каждая из строк 1–3 и 8–11 выполняется в течение фиксированного времени; длительность циклов **for** в строках 4–7 равна $\Theta(n_1 + n_2) = \Theta(n)$,[7] а в цикле **for** в строках 12–17 выполняется n итераций, на каждую из которых затрачивается фиксированное время.

Теперь процедуру MERGE можно использовать в качестве подпрограммы в алгоритме сортировки слиянием. Процедура MERGE_SORT(A, p, r) выполняет сортировку элементов в подмассиве $A[p..r]$. Если справедливо неравенство $p \geqslant r$, то в этом подмассиве содержится не более одного элемента, и, таким образом, он отсортирован. В противном случае производится разбиение, в ходе которого

[7] В главе 3 будет показано, как формально интерпретируются уравнения с Θ-обозначениями.

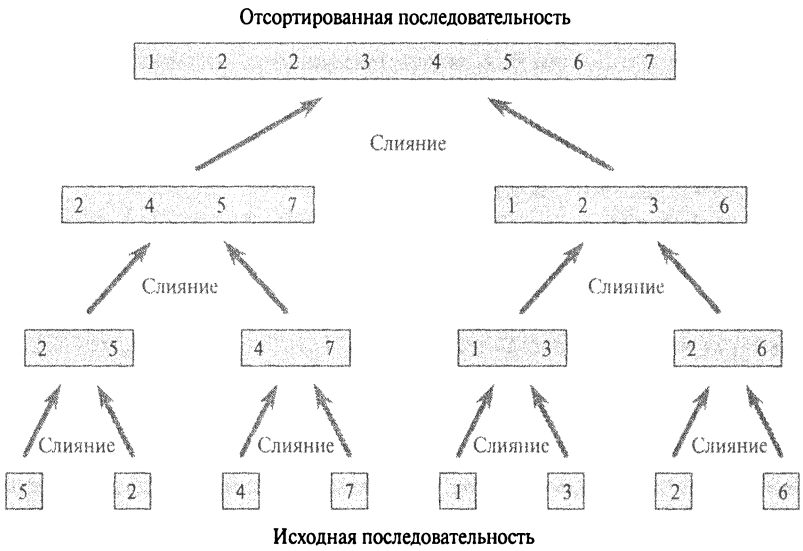

Рис. 2.4. Процесс сортировки массива $A = \langle 5, 2, 4, 7, 1, 3, 2, 6 \rangle$ методом слияния. Длины подлежащих объединению отсортированных подпоследовательностей возрастают по мере работы алгоритма

вычисляется индекс q, разделяющий массив $A[p..r]$ на два подмассива: $A[p..q]$ с $\lceil n/2 \rceil$ элементами и $A[q..r]$ с $\lfloor n/2 \rfloor$ элементами[8].

```
MERGE_SORT(A, p, r)
if p < r
   then q ← ⌊(p + r)/2⌋
        MERGE_SORT(A, p, q)
        MERGE_SORT(A, q + 1, r)
        MERGE(A, p, q, r)
```

Чтобы отсортировать последовательность $A = \langle A[1], A[2], \ldots, A[n] \rangle$, вызывается процедура MERGE_SORT(A, 1, *length* $[A]$), где *length* $[A] = n$. На рис. 2.4 проиллюстрирована работа этой процедуры в восходящем направлении, если n — это степень двойки. В ходе работы алгоритма происходит попарное объединение одноэлементных последовательностей в отсортированные последовательности длины 2, затем — попарное объединение двухэлементных последовательностей в отсортированные последовательности длины 4 и т.д., пока не будут

[8]Выражение $\lceil x \rceil$ обозначает наименьшее целое число, которое больше или равно x, а выражение $\lfloor x \rfloor$ — наибольшее целое число, которое меньше или равно x. Эти обозначения вводятся в главе 3. Чтобы убедиться в том, что в результате присваивания переменной q значения $\lfloor (p+r)/2 \rfloor$ длины подмассивов $A[p..q]$ и $A[q+1..r]$ получаются равными $\lceil n/2 \rceil$ и $\lfloor n/2 \rfloor$, достаточно проверить четыре возможных случая, в которых каждое из чисел p и r либо четное, либо нечетное.

получены две последовательности, состоящие из $n/2$ элементов, которые объединяются в конечную отсортированную последовательность длины n.

2.3.2 Анализ алгоритмов, основанных на принципе "разделяй и властвуй"

Если алгоритм рекурсивно обращается к самому себе, время его работы часто описывается с помощью ***рекуррентного уравнения***, или ***рекуррентного соотношения***, в котором полное время, требуемое для решения всей задачи с объемом ввода n, выражается через время решения вспомогательных подзадач. Затем данное рекуррентное уравнение решается с помощью определенных математических методов, и устанавливаются границы производительности алгоритма.

Получение рекуррентного соотношения для времени работы алгоритма, основанного на принципе "разделяй и властвуй", базируется на трех этапах, соответствующих парадигме этого принципа. Обозначим через $T(n)$ время решения задачи, размер которой равен n. Если размер задачи достаточно мал, скажем, $n \leqslant c$, где c — некоторая заранее известная константа, то задача решается непосредственно в течение определенного фиксированного времени, которое мы обозначим через $\Theta(1)$. Предположим, что наша задача делится на a подзадач, объем каждой из которых равен $1/b$ от объема исходной задачи. (В алгоритме сортировки методом слияния числа a и b были равны 2, однако нам предстоит ознакомиться со многими алгоритмами разбиения, в которых $a \neq b$.) Если разбиение задачи на вспомогательные подзадачи происходит в течение времени $D(n)$, а объединение решений подзадач в решение исходной задачи — в течение времени $C(n)$, то мы получим такое рекуррентное соотношение:

$$T(n) = \begin{cases} \Theta(1) & \text{при } n \leqslant c, \\ aT(n/b) + D(n) + C(n) & \text{в противном случае.} \end{cases}$$

В главе 4 будет показано, как решаются рекуррентные соотношения такого вида.

Анализ алгоритма сортировки слиянием

Псевдокод MERGE_SORT корректно работает для произвольного (в том числе и нечетного) количества сортируемых элементов. Однако если количество элементов в исходной задаче равно степени двойки, то анализ рекуррентного уравнения упрощается. В этом случае на каждом шаге деления будут получены две подпоследовательности, размер которых точно равен $n/2$. В главе 4 будет показано, что это предположение не влияет на порядок роста, полученный в результате решения рекуррентного уравнения.

Чтобы получить рекуррентное уравнение для верхней оценки времени работы $T(n)$ алгоритма, выполняющего сортировку n чисел методом слияния, будем рассуждать следующим образом. Сортировка одного элемента методом слияния длится в течение фиксированного времени. Если $n > 1$, время работы распределяется таким образом.

Разбиение. В ходе разбиения определяется, где находится средина подмассива. Эта операция длится фиксированное время, поэтому $D(n) = \Theta(1)$.

Покорение. Рекурсивно решаются две подзадачи, объем каждой из которых составляет $n/2$. Время решения этих подзадач равно $2T(n/2)$.

Комбинирование. Как уже упоминалось, процедура MERGE в n-элементном подмассиве выполняется в течение времени $\Theta(n)$, поэтому $C(n) = \Theta(n)$.

Сложив функции $D(n)$ и $C(n)$, получим сумму величин $\Theta(n)$ и $\Theta(1)$, которая является линейной функцией от n, т.е. $\Theta(n)$. Прибавляя к этой величине слагаемое $2T(n/2)$, соответствующее этапу "покорения", получим рекуррентное соотношение для времени работы $T(n)$ алгоритма сортировки по методу слияния в наихудшем случае:

$$T(n) = \begin{cases} \Theta(1) & \text{при } n = 1, \\ 2T(n/2) + \Theta(n) & \text{при } n > 1. \end{cases} \tag{2.1}$$

В главе 4 мы ознакомимся с теоремой, с помощью которой можно показать, что величина $T(n)$ представляет собой $\Theta(n \lg n)$, где $\lg n$ обозначает $\lg_2 n$. Поскольку логарифмическая функция растет медленнее, чем линейная, то для достаточно большого количества входных элементов производительность алгоритма сортировки методом слияния, время работы которого равно $\Theta(n \lg n)$, превзойдет производительность алгоритма сортировки методом вставок, время работы которого в наихудшем случае равно $\Theta(n^2)$.

Правда, можно и без упомянутой теоремы интуитивно понять, что решением рекуррентного соотношения (2.1) является выражение $T(n) = \Theta(n \lg n)$. Перепишем уравнение (2.1) в таком виде:

$$T(n) = \begin{cases} c & \text{при } n = 1, \\ 2T(n/2) + cn & \text{при } n > 1, \end{cases} \tag{2.2}$$

где константа c обозначает время, которое требуется для решения задачи? размер который равен 1, а также удельное (приходящееся на один элемент) время, требуемое для разделения и сочетания[9].

[9]Маловероятно, чтобы одна и та же константа представляла и время, необходимое для решения задачи, размер который равен 1, и приходящееся на один элемент время, в течение которого выполняются этапы разбиения и объединения. Чтобы обойти эту проблему, достаточно предположить, что c — максимальный из перечисленных промежутков времени. В таком случае мы получим верхнюю границу времени работы алгоритма. Если же в качестве c выбрать наименьший из всех перечисленных промежутков времени, то в результате решения рекуррентного соотношения получим нижнюю границу времени работы алгоритма. Принимая во внимание, что обе границы имеют порядок $n \lg n$, делаем вывод, что время работы алгоритма ведет себя, как $\Theta(n \lg n)$.

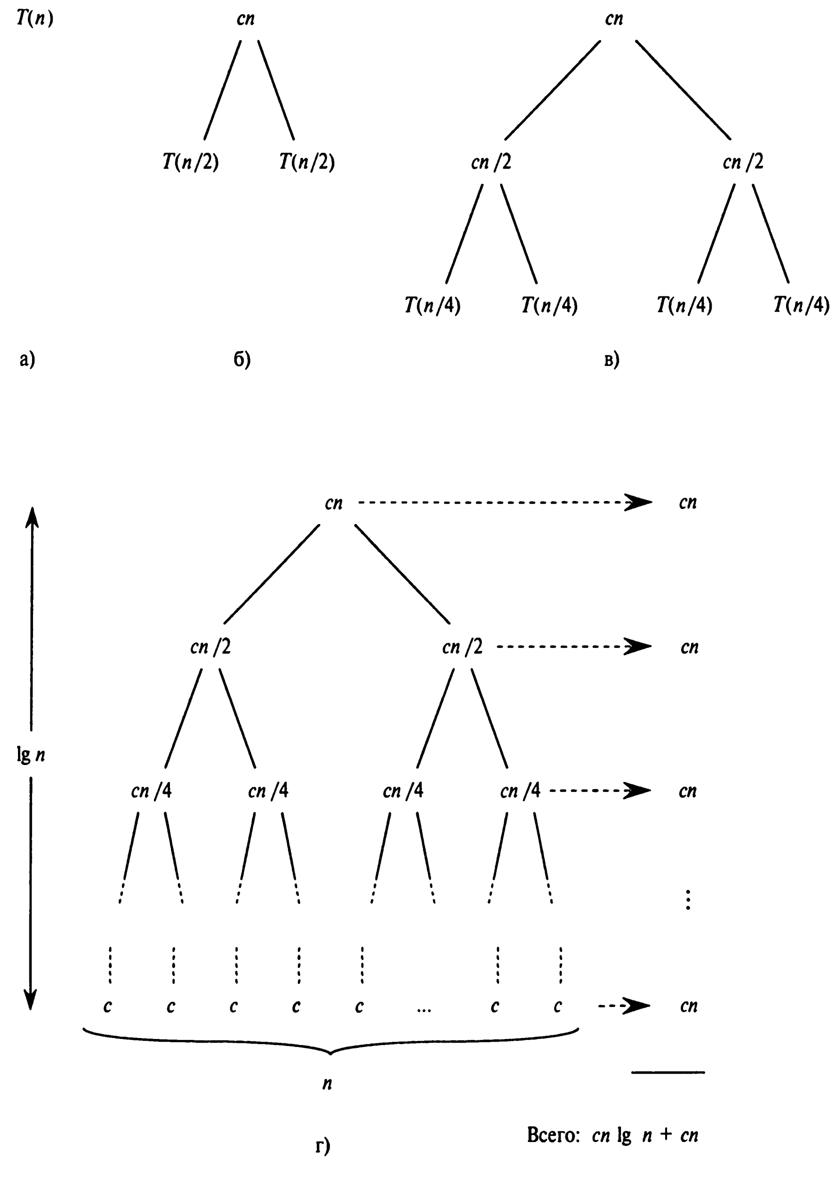

Рис. 2.5. Построение дерева рекурсии для уравнения $T(n) = 2T(n/2) + cn$

Процесс решения рекуррентного соотношения (2.2) проиллюстрирован на рис. 2.5. Для удобства предположим, что n равно степени двойки. В части *а* упомянутого рисунка показано время $T(n)$, представленное в части *б* в виде эквивалентного дерева, которое представляет рекуррентное уравнение. Корнем этого дерева является слагаемое cn (стоимость верхнего уровня рекурсии), а два поддерева, берущих начало от корня, представляют две меньшие рекуррентные последовательности $T(n/2)$. В части *в* показан очередной шаг рекурсии. Время выполнения каждого из двух подузлов, находящихся на втором уровне рекурсии, равно $cn/2$. Далее продолжается разложение каждого узла, входящего в состав дерева, путем разбиения их на составные части, определенные в рекуррентной последовательности. Так происходит до тех пор, пока размер задачи не становится равным 1, а время ее выполнения — константе c. Получившееся в результате дерево показано в части *г*. Дерево состоит из $\lg n + 1$ уровней (т.е. его высота равна $\lg n$), а каждый уровень дает вклад в полное время работы, равный cn. Таким образом, полное время работы алгоритма равно $cn \lg n + cn$, что соответствует $\Theta(n \lg n)$.

После того как дерево построено, длительности выполнения всех его узлов суммируются по всем уровням. Полное время выполнения верхнего уровня равно cn, следующий уровень дает вклад, равный $c(n/2) + c(n/2) = cn$. Ту же величину вклада дают и все последующие уровни. В общем случае уровень i (если вести отсчет сверху) имеет 2^i узлов, каждый из которых дает вклад в общее время работы алгоритма, равный $c(n/2^i)$, поэтому полное время выполнения всех принадлежащих уровню узлов равно $2^i c(n/2^i) = cn$. На нижнем уровне имеется n узлов, каждый из которых дает вклад c, что в сумме дает время, равное cn.

Полное количество уровней дерева на рис. 2.5 равно $\lg n + 1$. Это легко понять из неформальных индуктивных рассуждений. В простейшем случае, когда $n = 1$, имеется всего один уровень. Поскольку $\lg 1 = 0$, выражение $\lg n + 1$ дает правильное количество уровней. Теперь в качестве индуктивного допущения примем, что количество уровней рекурсивного дерева с 2^i узлами равно $\lg 2^i + 1 = i + 1$ (так как для любого i выполняется соотношение $\lg 2^i = i$). Поскольку мы предположили, что количество входных элементов равняется степени двойки, то теперь нужно рассмотреть случай для 2^{i+1} элементов. Дерево с 2^{i+1} узлами имеет на один уровень больше, чем дерево с 2^i узлами, поэтому полное количество уровней равно $(i + 1) + 1 = \lg 2^{i+1} + 1$.

Чтобы найти полное время, являющееся решением рекуррентного соотношения (2.2), нужно просто сложить вклады от всех уровней. Всего имеется $\lg n + 1$ уровней, каждый из которых выполняется в течение времени cn, так что полное время равно $cn(\lg n + 1) = cn \lg n + cn$. Пренебрегая членами более низких порядков и константой c, в результате получаем $\Theta(n \lg n)$.

Упражнения

2.3-1. Используя в качестве образца рис. 2.4, проиллюстрируйте работу алгоритма сортировки методом слияний для массива $A = \langle 3, 41, 52, 26, 38, 57, 9, 49 \rangle$.